游易-程序主题分享合辑

+ 收藏专题

知识管理部 等 2022.06.07 15:37 6242 321 184个资源

汇总从101至今的游易征稿文章合辑。

客户端引擎中的多线程应用

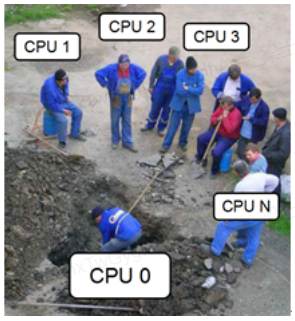
李先颖 2016.02.22 10:05 1750 9 4 查看原文

本文仅面向以下用户开放，请注意内容保密范围
查看权限：互娱正式-公开

介绍Messiah客户端引擎的多线程特性，分享一些使用C++进行客户端引擎多线程编程的小经验

0. 引言

从PC到移动设备，机器运算能力的提升一直都是支持着游戏不断进步和创新的一个基本原因。摩尔定律发展到现在，芯片工艺已越来越接近理论物理的极限，单个CPU算力的提升变得越发昂贵；近年来，厂商已开始通过横向增加处理器的数目来满足消费者对设备算力需求的持续增长，四核、八核的手机设备随处可见。然而目前的主流客户端游戏引擎，包括Unity和Unreal，大多都只能跑满一个或两个核，并不能充分挖掘出硬件的所有算力；不久的将来，如果我们拿出一部十几核甚至几十核的手机，打开一个依然只能单线程运行的游戏，体验将会是这样的：



天下事业部开发的Messiah引擎，从诞生之初就采用天然多线程的设计，在多核设备上有不错的表现。这篇文章将从多线程的同步模型和数据架构两方面来简单介绍Messiah引擎的多线程特性，并分享作者的一些使用C++进行客户端引擎多线程编程的小经验。

1. 同步模型

关于线程间的数据同步，最直接的方法就是对共享数据进行加锁保护。锁固然简单粗暴，然而锁也是昂贵的：无节制地使用锁会带来阻塞和繁的线程上下文切换，消耗大量性能。

为了干掉锁，Messiah使用一种叫做strand的模型来进行线程间的数据同步，这个概念来自于boost::asio：一个strand是一个任务链，它运行在一个线程池之上；发送到同一个strand中的各个任务一定能保证其执行时间片段没有重叠。例如下图中的A1,A2,A3,A4属于同一个strand，它们之间没有时间上的重叠；B1,B2也同属于另一个strand。

使用这一模型，我们可以将引擎的任务按照其所访问数据的亲缘性分类，然后把同一类的任务放到同一个strand中。这样一来，我们就不再需要对数据进行加锁，每个任务也可以流畅地执行下来，而不会阻塞。

Strand模型的本质是调度，用调度代替锁，可以极大地减少性能损耗。用城市交通作比喻，锁就像是司机各开各的，频繁变道，见缝插针；而strand则更像是一位高效的帮助疏通的交警。

当然，调度会带来一定程度上的延迟。为了不引起客户端的卡顿，我们将引擎中的任务分为了短任务和长任务两种，分别使用两个不同的线程池。短任务是指一帧之内能够执行完的，包括渲染和游戏逻辑相关；长任务则包括加载、shader编译、物理及地图模块的初始化工作，等等。

对于不涉及到全局共享数据的计算任务（包括骨骼动画Tick、粒子系统Tick、物理Tick、视锥裁剪、部分渲染算法等），我们允许其在短任务线程池的任一条线程上被调度执行，最大程度地保证引擎的并行度。

我们还会专门固定一条物理线程来提交图形API，并称之为设备线程。

2. 数据架构

“共享是魔鬼”，对于多线程编程来说确是如此。任何同步模型都解决不了数据架构带来的瓶颈。还用城市交通作比喻，失败的数据架构就好比糟糕的城市规划：假如整个城市所有的人上下班高峰期都必经同一立交桥，怎么疏通都不管用。

Messiah将数据按照亲缘性分为了Object（逻辑）、Render（渲染）、Shader（生成和编译shader）、IO等多个strand；然而客户端游戏引擎的复杂性，使得无可避免会有跨strand通信的需求，Messiah在设计和处理上遵循以下几点：

1）制定时间规则

在时间上保证数据共享的安全性。例如，又例如每一帧我们会有一段时间把Object strand暂停，在这段时间内，所有其他strands就可以放心对逻辑数据进行读操作。

2）用传递和交换代替共享

用流水和双倍的内存来换效率。例如骨骼动画和粒子系统，我们使用了double buffer；引擎在Render strand和Device thread提交这一帧的buffer的时候，Object strand已经可以开始进行下一帧的计算了；通过在每一帧Object strand暂停的临界区交换buffer指针可以实现高效的跨strand数据通信。

3）线程封闭（Thread local）

并行算法尽量使用Thread local data。例如我们在设计并行渲染算法的时候，把所有的Drawcall分发到了各个线程来进行计算，每个线程都拥有自己的一份RenderContext对象，可以直接进行访问。

4）无锁（Lock-free）

引擎的底层关键模块大量使用无锁数据结构支持并发。

3. 一些经验

最后分享几点使用C++进行客户端引擎多线程编程的实际经验：

1）制定命名规则

函数名加上后缀用来标识在什么strand执行，例如_Callback_on_ot表示这是一个应该在Object strand执行的函数，而_Tick_on_par则表示这个函数可能在任何线程执行（注：ot = object thread, par = parallel）。

变量名加上简单的前缀来表示其亲缘性，如下划线开头的成员变量表示逻辑相关的（也就是Object strand）变量。因此，如果在_Tick_on_par函数中尝试对_Transform变量进行写操作，那就是一个很明显的bug；应该先把要写得值记录下来，在_PostTick_on_ot函数中再赋值给_Transform。

2）使用Lambda

使用C++11的Lambda，可以极大增强代码的可读性和书写代码的流畅性，如：

函数式编程其实是一种非常自然的思维方式，它会强制和帮助我们对复杂的逻辑进行解耦。

3）注意生命周期

任何涉及长任务的回调（例如资源加载），this指针要以弱指针的方式进行传递，回调时需要检查对象是否已被销毁。

4）防止死锁

防止死锁的最好方式就是不要手写任何等待语句。Messiah最早的版本使用信号量来实现每帧Object strand的暂停，在某些极端情况下可能发生死锁；后来我们干掉了信号量，改为在调度器层实现Suspend和Resume方法，支持strand的暂停。

本内容仅代表个人观点，不代表网易游戏，仅供内部分享传播，不允许以任何形式外泄，否则追究法律责任。

收藏 26 点赞 9 分享 用手机查看

赏
快来成为第一个打赏的人吧~

全部评论 4

请输入评论内容

还可以输入 500 个字

(可添加1个视频+5张图片) 匿名 评 论

最热 最新

Falco(叶力扎提·金俄斯)
4楼
2021-04-06 14:29 回复 0

匿名
3楼 感谢，让我对这个引擎的特点有了一些认识
2019-11-07 09:37 回复 0

匿名
2楼 再讲的细一点就好了
2017-07-31 17:04 回复 0

加载完毕,没有更多了